Recognising handwritten ZIP (PIN) codes.

[Johann Fernandes, Ayush Nath]
[johann.fernandes, ayush.nath]_ug18@ashoka.edu.in
Ashoka University
May 8, 2017.

1. Introduction

Our project is meant to implement a full pipeline for recognising a handwritten zip code and displaying the location on a map. We separate our project into two parts, the segmentation of an image into separate digits, followed by classifying them into their respective labels. This project is inspired by the many years' work of Yann LeCun in digit recognition.

2. Dataset

The main obstacles in this research was finding a dataset that had the right labels for any region proposal neural network. We are currently using the MNIST[1] handwritten digit database for the digit recognition which contains a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits are of size 28x28 and are size-normalised and centred in a fixed-size image.

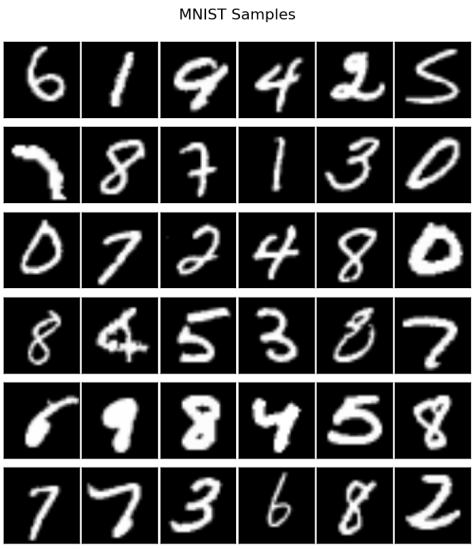


Figure 1

3. Approach

3.1. Image Segmentation

The image that we feed into the program consists of a handwritten ZIP code. The first step in our process is the segmentation of all the digits, so that each one can be passed through the classifier

[1] http://yann.lecun.com/exdb/mnist/

individually. For the same we input an image of size 168x28 the reason being the X axis has 6 digits to fit, each of length 28 pixels. This image was then cropped in a size of 28x28 and then sent to the estimator to predict each individual digit. The result of which was saved in an array and displayed as a 6 digit string.

3.2. Image Augmentation

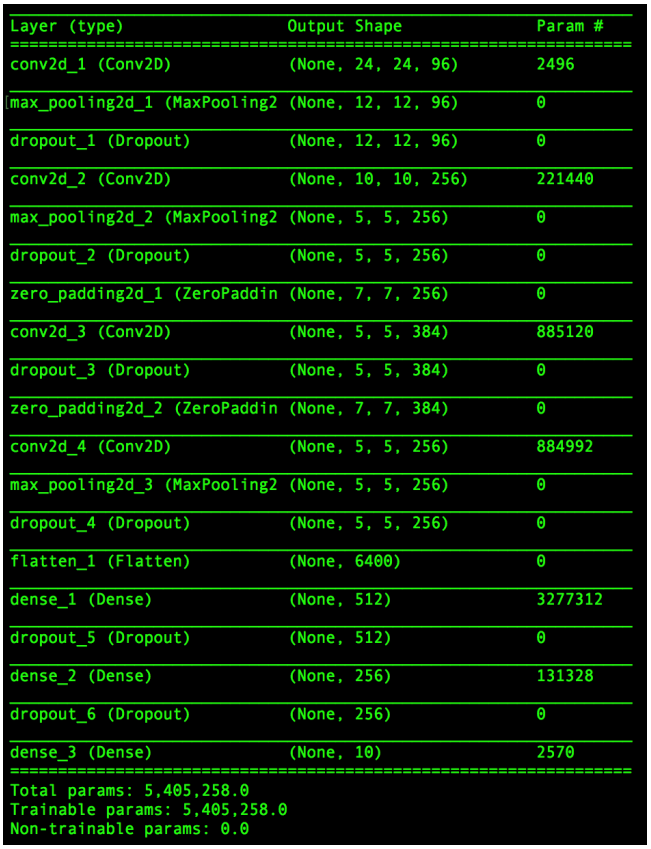
We augmented each of the 60,000 training samples that we got from the MNIST dataset, to create more images that were tilted, in a way that natural handwritten digits would look. We used the rotate_bound function from the imutils library. We rotated each image ± 15 degrees, every 5 degrees to create a dataset of 1,80,000 images. We were going to augment the images even further, until we realised that this was more than enough and we had started to achieve good accuracy.

3.3 Convolutional Neural Networks

Our project tackles the problem of character classification in natural handwritten images. We started solving the problem by using a Convolutional Neural Network consisting of three layers. To improve the performance of this architecture, we switched to the 16 layer VGG model with pre trained weights. Even after changing plenty of things we were not getting optimal results, and because of the large training time associated with such a large network, we decided to use transfer learning, and and implemented AlexNet with its pre-trained weight set. We realised that the hyper-parameters and dropout rates used in AlexNet were not optimal for our needs, so we decided to implement a similar, but simpler architecture, by making a few changes to the AlexNet model and training from scratch.

First, we changed the activation function of the last layer, from relu to softmax. This was better in our training, because we had 10 well defined labels. The sum of the probability of each class is always 1, so when the probability of one class increases, it drops for other classes. This was not necessarily the case with other functions. Before softmax was used, we achieved an accuracy of just 30% but after using softmax we started to have a breakthrough and our accuracy shot up to 99 percent. Second, we reduced the number of feature maps in each layer. The number of feature maps in AlexNet went from 96 to 4096. This was very unnecessary for our project, as our images contained only grayscale images, and not a lot of features. We also reduced the size of the kernel is most of the layers as we had to classify only 10 digits. Hence our kernel size ranged from (3,3) to

(5,5). Lastly, we removed the third layer. We saw that it was exactly the same as the fourth layer and not very different from other adjacent layers either. It was using expensive computational power and removing it helped us cut down on the training time as well. Our model and the parameters used can be seen in the following Figure.

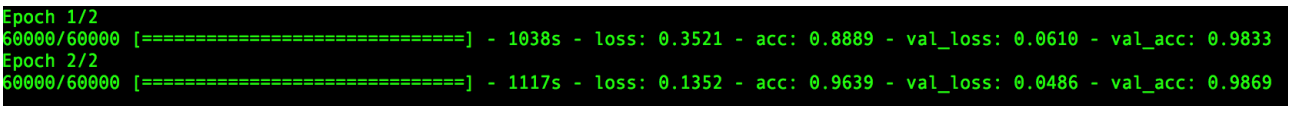


Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 24, 24, 96)	2496
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 96)	0
dropout_1 (Dropout)	(None, 12, 12, 96)	0
conv2d_2 (Conv2D)	(None, 10, 10, 256)	221440
max_pooling2d_2 (MaxPooling2	(None, 5, 5, 256)	0
dropout_2 (Dropout)	(None, 5, 5, 256)	0
zero_padding2d_1 (ZeroPaddin	(None, 7, 7, 256)	0
conv2d_3 (Conv2D)	(None, 5, 5, 384)	885120
dropout_3 (Dropout)	(None, 5, 5, 384)	0
zero_padding2d_2 (ZeroPaddin	(None, 7, 7, 384)	0
conv2d_4 (Conv2D)	(None, 5, 5, 256)	884992
max_pooling2d_3 (MaxPooling2	(None, 5, 5, 256)	0
dropout_4 (Dropout)	(None, 5, 5, 256)	0
flatten_1 (Flatten)	(None, 6400)	0
dense_1 (Dense)	(None, 512)	3277312
dropout_5 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131328
dropout_6 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 10)	2570

Total params: 5,405,258.0
Trainable params: 5,405,258.0
Non-trainable params: 0.0

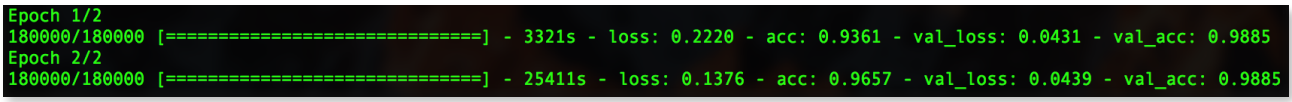
4. Expected Results

We trained our model three times with different training sizes and with different epochs, first with 60,000 images and later with 1,80,000 images after augmenting it. The results are shown in Figure 2, 3 and 4 respectively.



```
Epoch 1/2
60000/60000 [==============================] - 1038s - loss: 0.3521 - acc: 0.8889 - val_loss: 0.0610 - val_acc: 0.9833
Epoch 2/2
60000/60000 [==============================] - 1117s - loss: 0.1352 - acc: 0.9639 - val_loss: 0.0486 - val_acc: 0.9869
```

Figure 2



```
Epoch 1/2
180000/180000 [==============================] - 3321s - loss: 0.2220 - acc: 0.9361 - val_loss: 0.0431 - val_acc: 0.9885
Epoch 2/2
180000/180000 [==============================] - 25411s - loss: 0.1376 - acc: 0.9657 - val_loss: 0.0439 - val_acc: 0.9885
```

Figure 3

```
Epoch 1/10
180000/180000 [==============================] - 2870s - loss: 0.1501 - acc: 0.9653 - val_loss: 0.0513 - val_acc: 0.9863
Epoch 2/10
180000/180000 [==============================] - 3026s - loss: 0.1510 - acc: 0.9651 - val_loss: 0.0422 - val_acc: 0.9890
Epoch 3/10
180000/180000 [==============================] - 3121s - loss: 0.1533 - acc: 0.9653 - val_loss: 0.0461 - val_acc: 0.9885
Epoch 4/10
180000/180000 [==============================] - 3220s - loss: 0.1599 - acc: 0.9647 - val_loss: 0.0459 - val_acc: 0.9892
Epoch 5/10
180000/180000 [==============================] - 3258s - loss: 0.1687 - acc: 0.9643 - val_loss: 0.0394 - val_acc: 0.9903
Epoch 6/10
180000/180000 [==============================] - 3280s - loss: 0.1682 - acc: 0.9642 - val_loss: 0.0411 - val_acc: 0.9904
Epoch 7/10
180000/180000 [==============================] - 3338s - loss: 0.1773 - acc: 0.9642 - val_loss: 0.0482 - val_acc: 0.9888
Epoch 8/10
180000/180000 [==============================] - 3372s - loss: 0.1830 - acc: 0.9634 - val_loss: 0.0451 - val_acc: 0.9900
Epoch 9/10
180000/180000 [==============================] - 3470s - loss: 0.1865 - acc: 0.9626 - val_loss: 0.0421 - val_acc: 0.9918
Epoch 10/10
180000/180000 [==============================] - 3497s - loss: 0.2037 - acc: 0.9610 - val_loss: 0.0520 - val_acc: 0.9896
```

Figure 4

4.1 Testing our Model

To test our model, we considered the following examples.

1. We tested our model for two ‘1’s’ written in two different ways as seen in Figure 5 and 6 respectively. The predicted value for both of these digits was 1 by our model.

Figure 5

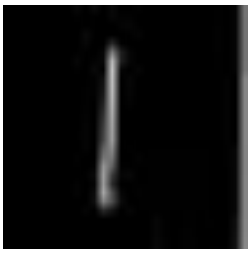


Figure 6

2. We decided to test an image with considerable noise. For the same, we drew a diagonal line across the digit to tamper with the pixel values of the digit. This was to consider for the fact that when we write a digit with pen and paper, the ink might not be uniform and hence will not give a perfect uniform digit. This can be seen in Figure 7. The predicted value for this was correct again as the model predicted 5.

Figure 7

3. We performed our last test on a sequence of digits as seen in a pin code. The following is shown in Figure 8, Figure 9 and Figure 10.

Figure 8

Figure 9

Figure 10

Observation: Our model predicted Figure 8 as the number i.e 123456. It predicted Figure 9 as 965320 but our model predicted Figure 10 as 917513 which means that it predicted two out of the six images wrong. This might be to account for the fact that the 6 does look like a 5 and the image would have some noise present in it.

5. Conclusion

The main take away of this project is that even after achieving an accuracy of 99%, one of the test cases predicted a wrong value. This is to show that very good neural networks also show incorrect information because of the noise present in the image. If a model is created to reduce or remove this noise from input images then the model can achieve really good results.